Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 739–745
DOI: 10.15439/2025F1909
ISSN 2300-5963 ACSIS, Vol. 43

Towards Game Level Generation Through LLM and GAN

Filip Martinović, Danijel Mlinarić, Juraj Dončević, Agneza Krajna, Ivica Botički
0009-0000-1347-4377, 0000-0002-5248-7223, 0000-0001-5221-6848, 0000-0001-8304-5463, 0000-0002-7378-3339
University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
{filip.martinovic, danijel.mlinaric, juraj.doncevic, agneza.krajna, ivica.boticki}@fer.hr

***Abstract*—This paper tackles the challenge of adaptive level generation in video games, focusing on generating content that aligns with player skill. A key limitation of procedural content generation (PCG) is achieving semantic control. Specifically, generating levels of varying difficulty with limited training data. To address this problem, we propose a hybrid approach combining Large Language Models (LLMs) and Generative Adversarial Networks (GANs). An LLM is used to generate a diverse, difficulty-labeled dataset of Snake game levels, which are validated with A* pathfinding to ensure playability. These levels serve as training data for GANs that are able to efficiently generate new levels. The system is evaluated through user study and playability metrics. Results show that the LLM-assigned difficulty labels correlate strongly with human perception. The achieved playability is 87% for easy levels and 36% for hard levels. Our findings demonstrate that the hybrid LLM-GAN approach enables scalable and semantically controlled content generation, balancing quality, adaptability, and computational efficiency.**

***Index Terms*—procedural content generation, generative adversarial networks, large language models, video games, playability, adaptive systems.**

I. Introduction

THE greatest advances in the software industry today are due to the ability to adapt to the personal needs and preferences of individual users. Functionalities such as Netflix's recommendation system or Duolingo's customizable lessons have given companies an edge over the competition. In recent years, new large language models (LLMs) have also been developed to answer users' questions. Similar principles are being explored in the development of video games, particularly through the automatic generation of content based on players' skills and behavior [1].

A key challenge in adaptive content generation is semantic control, i.e. the ability to create content with specific, interpretable characteristics (such as difficulty level) that align with human perception and intent, rather than producing statistically similar outputs. Traditional procedural content generation methods face challenges. They either rely on manually created rules that limit creativity or require extensive datasets that may not exist for specific domains. And even when successful, they often lack the semantic understanding to produce content with certain meaningful characteristics.

This paper continues research into adaptive software systems [2,3], extending from dynamic software updating to adaptive content generation. Using the game Snake as a controlled environment, we investigated three approaches for AI-driven level generation: standalone GANs, standalone LLMs and a novel hybrid LLM-GAN approach.

Our research specifically examines whether modern AI approaches can provide reliable semantic control for difficulty adaptation. Initial experiments with GANs trained on limited data (15 levels) demonstrated severe overfitting and poor generalization. Pure LLM generation showed excellent quality control through semantic understanding but proved computationally expensive for real-time applications. This motivated our hybrid approach: using LLMs to bootstrap diverse training datasets for GAN training, combining the semantic precision of language models with the computational efficiency of adversarial networks. We validated the semantic control capabilities through user study, demonstrating a strong correlation between LLM-assigned difficulty labels and human perception. The focus is to determine whether the proposed hybrid pipeline can achieve the benefits of both approaches while mitigating their individual limitations.

II. Related Work

The field of procedural content generation (PCG) for video games [4] has made significant progress through the application of artificial intelligence techniques. This section presents the main approaches to game level generation: traditional PCG methods and their scalability limitations, generative model applications, and playability assurance techniques that ensure generated content meets functional requirements.

A. Procedural Content Generation

Togelius et al. [5] define procedural content generation (PCG) as the algorithmic creation of game content with limited or indirect user input. Some games such as Minecraft[1] use

This work was co-funded by the European Union through the National Recovery and Resilience Plan (NPOO), under the project *AutoEvolve – Automated Evolution of Software*, project code: NPOO.C3.2.R3-I1.05.0073.

[1] https://www.minecraft.net/

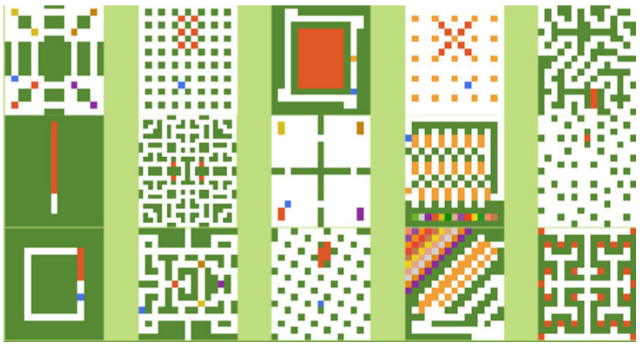

Fig. 1 Pre made levels for the Snake game. Green tiles represent walls, blue tiles represent the player's initial position, orange tiles represent box-es that can be moved by the player and all other colors of the tiles represent different types of food

PCG to create seemingly infinite worlds. Games in the roguelike and dungeon crawler genres, such as Spelunky[2], use it to create unpredictable levels that ensure each playthrough is unique. The challenge that arises from this approach is to manage the generation process to create meaningful and entertaining levels. This also requires substantial effort from the developer, who must set detailed restrictions to ensure playability.

B. Generative Models

Generative models are a class of machine learning algorithms designed to generate new data samples that resemble a given dataset. These models learn the underlying distribution of the data and can generate realistic outputs such as images, text, etc.

GANs [6] use the adversarial training process where the generator produces synthetic samples while the discriminator tries to distinguish real data from the generated ones. During the training process, the generator and discriminator play a mini-max game in which the goal for the generator is to produce realistic samples. Transformers [7], on the other hand, have introduced the mechanism of self-attention, which allows them to efficiently capture long-range dependencies. Although they are mainly used for text generation, they have also proven successful in image generation [8].

C. Generative Models for Level Creation

Recent research has investigated the use of machine learning (ML) for procedural content generation [9]. Generative Adversarial Networks (GANs) [6] have emerged as powerful tools for creating game levels that resemble human-designed content. As Volz et al. [10] have shown, GANs can capture the essence of existing Super Mario Bros levels while introducing new meaningful variations. Their approach uses a Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [11] to optimize the latent space of the network, allowing the creation of levels with certain desired properties, such as the wall density or the number of enemies. However, this method relies heavily on large datasets of existing levels and is constrained by the design patterns present in the original corpus, potentially limiting novelty.

The CESAGAN (Conditionally Embedded Self-Attention GAN) framework [12] builds on basic GAN architectures and incorporates explicit constraints into the generation process. This model encodes specific requirements, such as the number of walls or enemies, into feature vectors that guide the generation process. Bootstrapping techniques were used to solve the problem of a lack of data in the game-level datasets. This involves iteratively adding newly generated levels that meet the playability criteria, such as the correct number of tile types and the existence of a path to all goals, to the training dataset. This approach has significantly increased the percentage of playable levels while reducing the number of duplicates.

D. Playability

One of the main problems encountered with PCG is ensuring that created levels can be completed by the players. There are several approaches to solving this problem from different perspectives. One innovative approach combines classification with explainability techniques [13]. After identifying unplayable levels, models such as SHAP (SHapley Additive exPlanations) [14] or Integrated Gradients [15] assign importance values to each coordinate in the level, highlighting areas that are responsible for the unplayability. These values lead to targeted changes to problematic areas, while the overall structure and aesthetics of the level are preserved.

Zhang et al. [16] used GAN to generate Zelda levels and then used Mixed Integer Linear Programming (MILP) to repair unplayable levels. MILP provides a mathematical approach to level repair by representing levels as graphs in which the nodes correspond to tiles (walls, empty spaces, players, keys, doors or enemies) and the edges represent the connections between tiles. The playability problem is formulated as a network flow problem with sources (e.g. players), targets (e.g. doors, keys) and obstacles (e.g. walls). The solution minimizes the number of changes required to make a level playable while satisfying constraints related to flow preservation and obstacle avoidance. While this approach guarantees playability, the complexity of implementation could increase significantly for games with dynamic elements such as moving opponents.

Cooper and Sarkar [17] have used specialized repair agents to identify and repair unplayable areas. These agents can perform otherwise forbidden actions (e.g., walking through walls), but at a high cost. By identifying expensive actions on an agent's path to the goal, the system can make targeted modifications to the level, such as converting walls into empty spaces.

Jain et al. [18] used autoencoders [19] to generate new levels based on an existing corpus of Super Mario Bros levels. Their model compresses the level designs into a lower-dimensional latent space and then reconstructs them, effectively learning the basic patterns and structures that compose valid

[2] https://www.spelunkyworld.com/

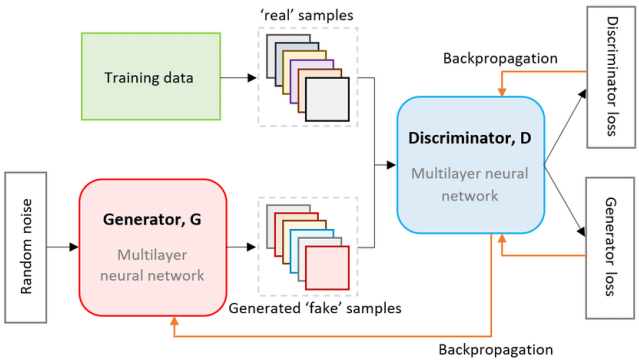

Fig. 2 Architecture of GAN [24]

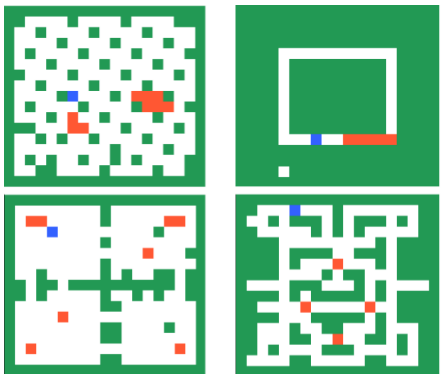

Fig. 3 Levels generated by GAN on initial dataset

levels. They have also used autoencoders to repair unplayable levels by running them through the model, which leverages its learned representations to reconstruct missing or problematic elements while maintaining overall coherence.

III. Hybrid Approach to Procedural Content Generation

This section describes our experimental framework for developing an adaptive Snake game that dynamically generates levels tailored to player skill. Our approach addresses a fundamental challenge in procedural content generation: balancing quality, controllability, and computational efficiency. While LLMs excel at generating semantically meaningful content with explicit difficulty control, they are computationally expensive for real-time applications. Conversely, once trained, GANs can efficiently generate content, but typically lack semantic understanding and require large datasets. Our hybrid pipeline combines these approaches by using LLMs to generate diverse, difficulty-labelled training data, then training specialized GAN models on this data for efficient generation at runtime. The system operates in two distinct phases. In the training phase, we use structured prompts to generate levels of controllable difficulty, validate their playability, and train separate GAN models for easy and hard difficulty levels.

A. Initial Dataset

One of the biggest challenges in training generative models is the lack of high-quality, diverse training data. The initial dataset was extracted from the 15 pre-made levels of the Google Snake Game Level Editor mod [20] (in Fig. 1).

However, this initial dataset of 15 levels presents a significant limitation for training GAN, which typically requires hundreds or thousands of training samples to learn meaningful data distributions and avoid overfitting. To address data scarcity, we used LLM to generate additional training data, as detailed in the section IV.A.

The pre-made levels consist of different types of tiles such as walls, crates, players and different types of food. To simplify the generation process, we have focused on 4 types of tiles that are essential for the game: walls, empty spaces, food and the player's starting position. All the different food types have been combined into one and all other tile types have been treated as empty spaces.

Individual pre-made level images are available on GitHub [21]. The images were processed by extracting tiles based on their colour. The levels can be displayed in different ways. Inspired by the game levels in The Video Game Levels Corpus (VGLC) [22], we opted for the 2D grid representation. A specific character symbol is used for each tile type. In our simplified levels, "W" stands for walls, "-" for empty areas, "F" for food and "S" for snake head, as shown in Fig. 4.

To encode the levels for training with GANs, each tile type is encoded in a separate channel, so that the levels were represented using four channels. Since the pre-made levels have different sizes, further preprocessing was required. In particular, all levels were padded with empty spaces up to a size of 32x32 tiles.

B. GAN Model

The architecture of the GAN model is based on the model from Arjovsky et al. [23]. In this design, the discriminator consists of blocks of strided convolutional layers followed by batch normalization layers and LeakyReLU activation layers. The generator uses transposed convolutional layers followed by batch normalization and ReLU layers. The last transposed convolutional layer is followed by the hyperbolic tangent function (*tanh*), which maps the outputs to the range [-1,1]. The architecture is adapted to 32x32 size images by removing the deepest layer of both the discriminator and the generator. The architecture of the entire system is shown in Fig. 2.

```
WWWWWWWWWWWWWWWWWWWW
W------WWWWW------W
W-A---S---------A-W
W--WW--WWWWW--WW--W
W--WW--WWWWW--WW--W
W----A-------A----W
W------WWWWW------W
WW-WW-WWWWWWW-WW-WW
WW-WW-WWWWWWW-WW-WW
WW-WW-WWWWWWW-WW-WW
W------WWWWW------W
W----A-------A----W
W--WW--WWWWW--WW--W
W--WW--WWWWW--WW--W
W-A-------------A-W
W------WWWWW------W
WWWWWWWWWWWWWWWWWWWW
```

Fig. 4 Example of pre-made level in 2D format

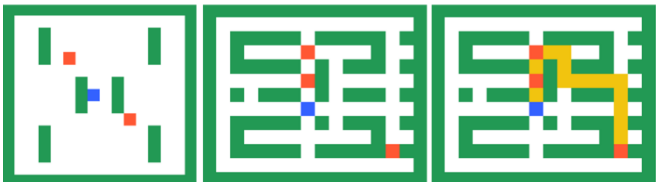

Fig. 5 a) Easy level generated by LLM, b) hard level generated by LLM and c) hard level generated by LLM with shortest path found with A* algorithm

After the training process, the generator can accurately replicate levels in the training dataset, as shown in Fig. 3. However, since only a limited amount of training data is available, the model tends to overfit. It replicates training samples instead of generating varied new levels. Furthermore, to generate levels with the desired difficulty, a larger, balanced dataset with difficulty labels is required.

C. LLM for Training Set

Instead of designing new levels manually, large language models (LLMs) can be used for this task. To generate new levels, the LLM must first be given clear instructions on how to generate them. It must also be given some levels as a reference. In most cases, text prompts are used as input to the model and the output is usually in the form of text. When generating levels with LLMs, we explored two primary output formats: 2D textual grids and structured representation. As level examples, we passed premade levels in 2D format, as shown in Fig. 4. The prompt to the model was as follows: "I have attached game levels for the game Snake. 'W' represents walls and '-' represents empty spaces. Generate more similar levels".

We found that the text outputs, especially for larger levels, often contained errors, such as an incorrect number of rows or an incorrect number of elements in a row. While most of these errors could be corrected manually, our goal is to automate the process. To generate high-quality, structured game layers, we combined GPT-4o and the OpenAI API Structured Outputs feature [25] that allowed us to capture the output in JSON format. To shape the whole process as a pipeline, we used the LangChain [26] framework in combination with a Pydantic-based output parser. Instead of generating free-form grids, we used pydantic.BaseModel to define a detailed schema to explicitly describe the components of a game level. The schema contains fields with the grid size (height and width of the level), a list of wall coordinates, the position of the snake head and a list of food coordinates. The output parser allowed us to validate each generated level and automatically discard levels that did not exactly match the schema. A typical prompt asked the model to generate a solvable Snake level of a given size with specific design constraints (the food must not be placed on the walls, the level must be solvable). The output was automatically parsed and validated. Inconsistencies or missing fields triggered errors, allowing us to discard or regenerate the sample in question. This approach provided a clean, normalized dataset suitable for conversion to structured tensors for GAN training. It also gave us the opportunity to generate new levels under certain conditions. The difficulty of the levels is particularly interesting. With carefully designed prompts, it is possible to generate easier or harder levels, as can be seen in Fig. 5. Easy levels have fewer walls, and food is generated closer to snake's head, while harder levels have more complex wall structures and food is harder to reach. In some cases, however, there is no clear difference in the difficulty of the generated levels.

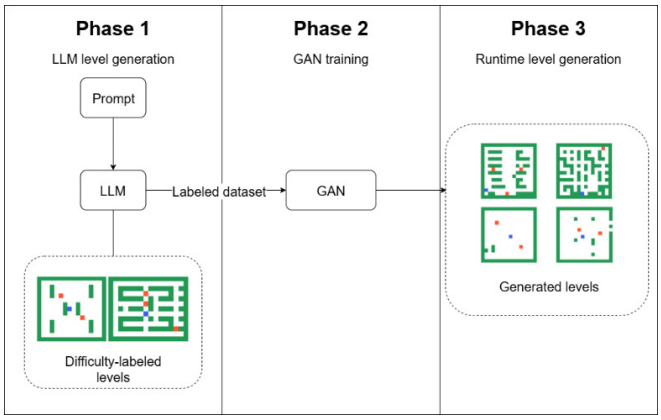

Fig. 6 System architecture of the hybrid LLM-GAN level generation pipeline

Even though there are multiple methods that can check the playability and even repair levels (section II.C), we only calculated the shortest path from the snake's head to all the food items. It can be calculated using the A* algorithm [27], adapted for multiple goals, where additional movement constraints apply: the snake cannot immediately turn back (return to the direction it just came from). The snake's ability to grow incrementally was not considered and it was treated as a single tile. The Manhattan distance was used as a heuristic function. Knowing the shortest path allows comparison of levels from the unbiased perspective of a "perfect player". We track metrics such as path length, the number of turns, the number of tight turns (around walls), and the longest straight path. We also track metrics related to the level itself, such as the percentage of walls, the number of dead ends, the number of "chokepoints" (narrow passages with walls on both sides). Regardless of these metrics, the difficulty depends significantly on the placement of the food.

IV. Experiment Setup

To create a substantially larger dataset for GAN training and validate our hybrid approach, we employed a comprehensive three-phase experimental framework. This methodology enables systematic comparison of the cost and quality of pure LLM generation, the effectiveness of hybrid GAN training, and the validation of semantic control through a user study, providing quantitative analysis of the trade-offs between the different approaches. The system architecture is shown in Fig. 6.

A. Generating Dataset for GAN Training

We used levels generated by an LLM in text format, as described in section III.A, and input them to LLM as examples. When selecting the examples, focus was to ensure that there

TABLE I

MODEL COMPARISON BETWEEN LEVEL DIFFICULTY

Model *Difficulty*	**Max (Avg) Playability Rate (PR)**	**Computational cost**	**Generation time**	**Semantic control**
GAN *Hard*	36 (24*) %	2h training duration	< 1s	Limited
GAN *Easy*	87 (74*) %	2h training duration	< 1s	Limited
LLM *Hard*	66 (100**) %	~705 tokens per level	~30s	Excellent
LLM *Easy*	100 %	~412 tokens per level	~30s	Excellent

*Average of top 5 hyperparameter configurations
**After validation and regeneration of failed levels

is a clear difference between hard and easy levels, like more open spaces and food closer to the player for easier levels, and more narrow passages and food placed in more difficult places for harder levels.

We then generated 50 easy and 50 difficult levels with structured few-shot prompts. Each generation started with a prompt with 3569 tokens for easy levels and 8495 tokens for difficult levels. The model's responses varied in complexity, averaging 412 tokens for easy levels and 705 tokens for difficult levels. Due to the nature of generative models that not all levels were viable. Some were unplayable or logically broken and had to be discarded, resulting in a higher number of tokens spent. As this process immediately generated 50 playable easy levels, there were 17 unplayable hard levels that we discarded. The total number of tokens used to generate all 100 levels was 617131. We can see that the generation process, while successful, consumes a lot of resources and is not practical when a big set of levels is needed.

B. GAN Training

With the generated dataset, we trained two separate GAN models: one for easy and one for difficult levels. Each model was trained[3] with a corresponding subset of 50 levels.

All levels were pre-processed as described in III.A. The training data was normalized to the range [-1,1] to align with the *tanh* activation in the generator's output layer, as mentioned in section III.B.

We explored different values for the learning rate, the stack size, the dimension of the latent vector, and the number of times the discriminator is trained before training the generator during each epoch. The playability rate (PR) was used as the optimization target. It is defined as number of solvable levels per 100 generated levels. The models were trained for 2000 epochs.

C. User Study for Semantic Control Validation

To evaluate the perceived difficulty of the LLM generated game levels, we conducted a user study involving seven participants. Each participant was asked to rate 100 level

[3]Training was conducted on NVIDIA GeForce GTX 1050 for approximately 4 hours. We used Optuna framework [28] for hyperparameters and monitored convergence through playability rate (PR) metric.

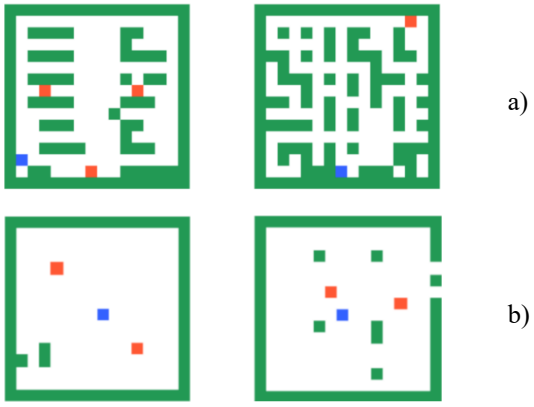

Fig. 7 Playable levels generated by GAN models trained on: a) "hard" dataset, b) "easy" dataset

images on a scale from 1 (very easy) to 5 (very difficult). The dataset consisted of 50 "easy" and 50 "hard" levels generated by a large language model in IV.A. However, users were not informed of these labels. The difficulty levels were randomly shuffled and presented to all users in the same order to ensure consistency. After collecting the individual ratings, we calculated the average rating for each level and analyzed its Pearson correlation with various structural features extracted from the levels described in III.C. This enabled us to identify which features most strongly influence the difficulty perceived by the users and the correspondence between the difficulty assigned as input to the LLM and the user ratings.

V. RESULTS

We present our findings in two main parts: first, validation of LLM semantic control through a user study, showing a strong correlation between assigned and perceived difficulty; second, the analysis of GAN performance when trained on LLM-generated datasets, revealing significant differences between easy and hard level generation complexity.

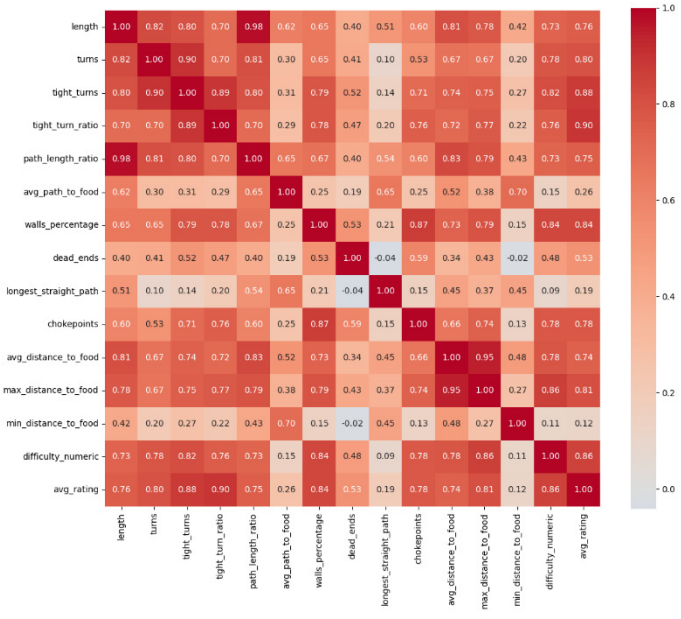

Fig. 8 Correlation matrix between structural level features and user difficulty ratings (n=7 participants, 100 levels)

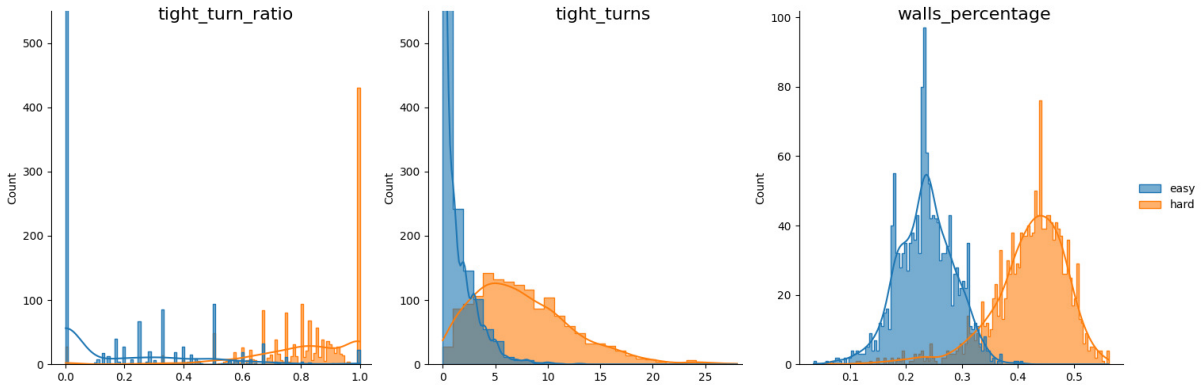

Fig. 9 Distribution comparison of key metrics between GAN-generated “easy” and “hard" levels

A. LLM Generated Levels

Based on the correlation (Fig. 8), we have identified several features that exhibit a strong relationship with user perceived difficulty. The most prominent is the *tight turn ratio*, which has the highest positive correlation with average user ratings, suggesting that levels requiring frequent sharp changes of direction are consistently rated as more difficult. Other features such as *total number of turns* and *percentage of walls* also correlate strongly with higher difficulty ratings, suggesting that spatial complexity and environmental constraints play a central role in perceived challenge. However, several features are highly intercorrelated, most notably *tight turns* and *turns*, as well as *average* and *maximum distance to food*, leading to redundancy and suggesting that a more compact feature set could be used without significant information loss. Interestingly, the binary *difficulty label* originally assigned to the level generation prompt ("easy" vs. "hard") shows a strong correlation with the actual user ratings.

Results show that the generated levels are highly consistent with players' perceptions, which strengthens confidence in the quality of the generated levels and can serve as a useful basis for further automation. However, a larger-scale study would be needed to confirm these findings across diverse player populations. It is important to note that our analysis is based on correlation rather than causation, further controlled experiments are needed to confirm which features directly influence perceived difficulty.

B. GAN Generated Levels

We have selected the five best performing hyperparameter configurations for each model (Easy GAN and Hard GAN) based on the playability rate (PR). For each of these configurations, we also identified the top three training checkpoints that produced the highest quality outputs. From each of these checkpoints, we generated 100 playable level samples, resulting in a diverse evaluation set. We then computed metrics described in IV.C for these generated levels. As shown in Table I, the model trained on the “hard” dataset generates much less playable levels than the one trained on the easy dataset. The significant difference in playability rates (87% vs 36%) suggests that difficult levels require more complex spatial reasoning that our current GAN architecture struggles to capture. This indicates that difficulty affects not only player perception but also computational generation complexity. Similar to the results in A, some features, such as number of *tight turns*, *tight turn ratio* and *percentage of walls*, show a clear difference between the generated easy and hard levels[4], as shown in Fig. 9. Models are able to generate completely new playable levels with distinct features as shown in Fig. 7.

VI. Conclusion

Our research explored different approaches to procedural content generation (PCG) of game level s. We found that, while efficient for generating a large number of levels, GANs require big and quality datasets. LLMs, on the other hand, produce high quality levels with very little input, but require a lot of resources to generate a larger amount of levels. We proposed a hybrid approach by combining the flexibility of large language models (LLMs) with the efficiency of generative adversarial networks (GANs). LLMs proved effective in generating structured and semantically rich level layouts, particularly when guided by schema-constrained prompting. However, due to high computational costs, LLMs are best suited for generating curated datasets. In contrast, GANs, once trained, can quickly generate large volumes of content but exhibit limited generalization when data is scarce. Our user study confirmed that LLM-labeled difficulty levels align well with player perceptions, and that certain structural features (e.g., tight turns, wall density) correlate strongly with perceived difficulty. These insights can guide future systems toward real-time, skill-adaptive content generation.

References

[1] G. N. Yannakakis and J. Togelius, ‘Experience-Driven Procedural Content Generation’, *IEEE Trans. Affect. Comput.*, vol. 2, no. 3, pp. 147–161, Jul. 2011, doi: 10.1109/T-AFFC.2011.6.

[2] D. Mlinarić, ‘Extension of dynamic software update model for class hierarchy changes and run-time phenomena detection’, info:eu-repo/semantics/doctoralThesis, University of Zagreb. Faculty of Electrical Engineering and Computing. Department of Applied

[4]Complete results can be found at: https://fer-autoevolve.github.io/results/

Computing, 2020. Accessed: May 26, 2025. [Online]. Available: https://urn.nsk.hr/urn:nbn:hr:168:087042

[3] D. Mlinarić, J. Dončević, M. Brčić, and I. Botički, 'Revolutionizing Software Development: Autonomous Software Evolution', in 2024 47th MIPRO ICT and Electronics Convention (MIPRO), May 2024, pp. 224–228. doi: 10.1109/MIPRO60963.2024.10569871.

[4] N. Shaker, J. Togelius, and M. J. Nelson, Procedural Content Generation in Games. in Computational Synthesis and Creative Systems. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-42716-4.

[5] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, 'What is procedural content generation? Mario on the borderline', in *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, in PCGames '11. New York, NY, USA: Association for Computing Machinery, Jun. 2011, pp. 1–6. doi: 10.1145/2000919.2000922.

[6] I. J. Goodfellow *et al.*, 'Generative Adversarial Networks', Jun. 10, 2014, *arXiv*: arXiv:1406.2661. doi: 10.48550/arXiv.1406.2661.

[7] A. Vaswani et al., 'Attention Is All You Need', Jun. 12, 2017, arXiv: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.

[8] A. Dosovitskiy *et al.*, 'An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale', Oct. 22, 2020, *arXiv*: arXiv:2010.11929. doi: 10.48550/arXiv.2010.11929.

[9] A. Summerville *et al.*, 'Procedural Content Generation via Machine Learning (PCGML)', *IEEE Trans. Games*, vol. 10, no. 3, pp. 257–270, Sep. 2018, doi: 10.1109/TG.2018.2846639.

[10] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, 'Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network', May 02, 2018, *arXiv*: arXiv:1805.00728. doi: 10.48550/arXiv.1805.00728.

[11] N. Hansen, S. D. Müller, and P. Koumoutsakos, 'Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)', *Evol. Comput.*, vol. 11, no. 1, pp. 1–18, Mar. 2003, doi: 10.1162/106365603321828970.

[12] R. Rodriguez Torrado, A. Khalifa, M. Cerny Green, N. Justesen, S. Risi, and J. Togelius, 'Bootstrapping Conditional GANs for Video Game Level Generation', in *2020 IEEE Conference on Games (CoG)*, Aug. 2020, pp. 41–48. doi: 10.1109/CoG47356.2020.9231576.

[13] M. Bazzaz and S. Cooper, 'Guided Game Level Repair via Explainable AI', Nov. 04, 2024, *arXiv*: arXiv:2410.23101. doi: 10.48550/arXiv.2410.23101.

[14] S. M. Lundberg and S.-I. Lee, 'A Unified Approach to Interpreting Model Predictions', in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. Accessed: Apr. 02, 2025. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

[15] N. Kokhlikyan *et al.*, 'Captum: A unified and generic model interpretability library for PyTorch', Sep. 16, 2020, *arXiv*: arXiv:2009.07896. doi: 10.48550/arXiv.2009.07896.

[16] H. Zhang, M. C. Fontaine, A. K. Hoover, J. Togelius, B. Dilkina, and S. Nikolaidis, 'Video Game Level Repair via Mixed Integer Linear Programming', Oct. 13, 2020, *arXiv*: arXiv:2010.06627. doi: 10.48550/arXiv.2010.06627.

[17] S. Cooper and A. Sarkar, 'Pathfinding Agents for Platformer Level Repair'.

[18] R. Jain, A. Isaksen, and C. Holmg, 'Autoencoders for Level Generation, Repair, and Recognition'.

[19] G. E. Hinton and R. R. Salakhutdinov, 'Reducing the Dimensionality of Data with Neural Networks', *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006, doi: 10.1126/science.1127647.

[20] 'Google Snake Mods'. Accessed: Apr. 04, 2025. [Online]. Available: https://googlesnakemods.com/v/4/

[21] 'DarkSnakeGang/GoogleSnakeLevelEditor: Level Editor Mod for Google Snake'. Accessed: Apr. 08, 2025. [Online]. Available: https://github.com/DarkSnakeGang/GoogleSnakeLevelEditor

[22] A. J. Summerville, S. Snodgrass, M. Mateas, and S. Ontañón, 'The VGLC: The Video Game Level Corpus', Jul. 03, 2016, *arXiv*: arXiv:1606.07487. doi: 10.48550/arXiv.1606.07487.

[23] M. Arjovsky, S. Chintala, and L. Bottou, 'Wasserstein GAN', Dec. 06, 2017, *arXiv*: arXiv:1701.07875. doi: 10.48550/arXiv.1701.07875.

[24] C. Little, M. Elliot, R. Allmendinger, and S. S. Samani, 'Generative Adversarial Networks for Synthetic Data Generation: A Comparative Study', Dec. 03, 2021, arXiv: arXiv:2112.01925. doi: 10.48550/arXiv.2112.01925.

[25] 'Structured Outputs - OpenAI API'. Accessed: May 24, 2025. [Online]. Available: https://platform.openai.com

[26] 'LangChain'. Accessed: May 22, 2025. [Online]. Available: https://www.langchain.com/langchain

[27] P. E. Hart, N. J. Nilsson, and B. Raphael, 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths', *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968, doi: 10.1109/TSSC.1968.300136.

[28] 'Optuna - A hyperparameter optimization framework', Optuna. Accessed: May 23, 2025. [Online]. Available: https://optuna.org/